

REMARKS

Claims 1-32 are pending in the present application. By this Response, claims 2, 3, 5-7, 12, 13, 16-22, 27 and 28 are amended for clarity of the subject matter being claimed. Reconsideration of the claims in view of the above amendments and the following remarks is respectfully requested.

Amendments were made to the specification to correct errors and to clarify the specification. No new matter has been added by any of the amendments to the specification.

I. 35 U.S.C. § 103, Alleged Obviousness, Claims 1-4, 7, 8, 16-19, 22, 23, and 30

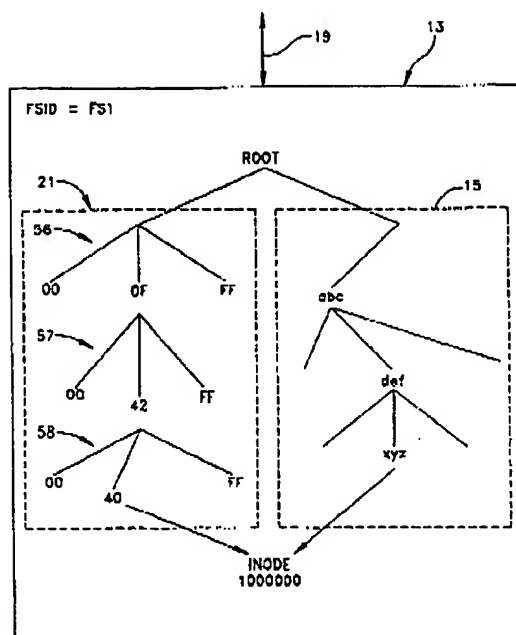
The Final Office Action rejects claims 1-4, 7, 8, 16-19, 22, 23, and 31 under 35 U.S.C. § 103(a) as being unpatentable over Kao (U.S. Patent No. 5,870,734) in view of Pinkoski (U.S. Patent No. 5,742,817). This rejection is respectfully traversed. Because this rejection is essentially the same as in the previous Office Action, this rejection is respectfully traversed for the same reasons stated in the previous response filed August 13, 2004, the remarks of which are hereby incorporated by reference. In the previous Response, Applicants asserted that (a) neither Kao nor Pinkoski discusses a file system that has been moved and (b) neither of the references teaches or suggests using information found in the referencing object to look up the location of the file system in a file system location database. All limitations of the claimed invention must be considered when determining patentability. In re Lowry, 32 F.3d 1579, 1582, 32 U.S.P.Q.2d 1031, 1034 (Fed. Cir. 1994). In comparing Kao and Pinkoski to the claimed invention to determine obviousness, limitations of the presently claimed invention may not be ignored. The present invention in claim 1 recites "receiving a request for a referencing object from a client, wherein the referencing object refers to a referenced file system that has been moved to a location on a different server." Such a feature is not taught or suggested by Kao and Pinkoski, either alone or in combination. Therefore, claim 1 is not obvious in view of Kao and Pinkoski. The following remarks are provided in rebuttal to the statements in the Final Office Action beginning on page 2, section 3.

The Final Office Action states:

Applicant's argued that, "Kao and Pinkoski do not discuss a file system that has been moved (Page 12, lines 36- and Page 13, lines 28-30).

In combination, Kao and Pinkoski teach transferring the request of an operation over the communications link to the different servers (Pinkoski – as shown in fig.2 and col. 4, lines 8-28) and using the message to locating the file in the different path (abstract, col. 3, lines 12-22 and col. 5, lines 1-30).

Applicants respectfully submit that the arguments presented in the August 13, 2004 response clearly points out that Kao and Pinkoski, taken alone or in combination, fails to teach or suggest **"receiving a request for a referencing object from a client, wherein the referencing object refers to a referenced file system that has been moved to a location on a different server"** (emphasis added). In the August 13, 2004 Response, Applicants stated Kao and Pinkoski do not discuss a file system that has been moved. **Figure 6A** of the present application shows client 608 first requests file X on Server Number 1 606, but after being redirected, client 608 is able to request file X from Server 2 616, where it now resides. The prior Office Action acknowledges that Kao does not clearly teach returning a redirection message indicating the location of the referenced file system to the client. However, the Office Action stated that Pinkoski teaches the transferring message indicating the existence of the alternate path name to be returned to the client or requester of the system. In response, Applicants cited **Figure 4** and column 3, lines 10-21, which are as follows:



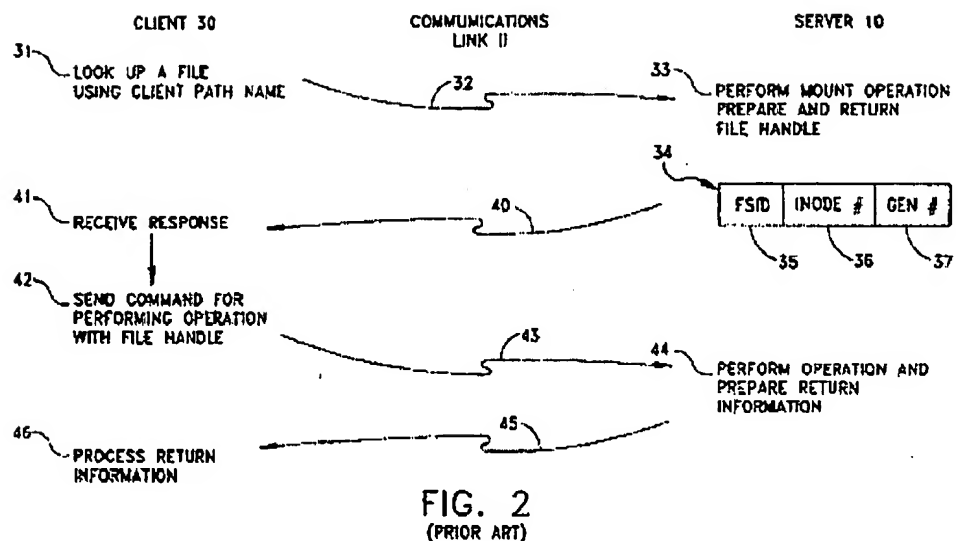
(Figure 4)

This invention is applicable to a file system in which client path names and unique numerical values identify each file and in which certain server operations return to a client in the network a file handle including the unique numerical value. The file system includes a first set of links for establishing correspondences between the client path names and unique numerical values. In accordance with this invention, a server uses the file handle during subsequent operations by converting the unique numerical value into an alternate path name by means of a second set of links for establishing correspondences between the alternate path names and the locations of the files in the file system.

(Column 3, lines 10-21)

In this figure and section, Pinkoski discloses an alternate route to reach a file that remains in the same place. While this alternate pathway may be shorter than the original pathway, the end location is the same.

The present Office Action states that Kao and Pinkoski teach transferring the request of an operation over the communications link to the different servers and using the message to locate the file in the different path at Figure 2, Column 4, lines 8-28, the abstract, column 3, lines 12-22 (shown above) and column 5, lines 1-30 which reads as follows:



(Figure 2)

FIG. 2 depicts, in general form, the process by which a client 30 receives a file handle from the server 10 over the communications link 11 using prior art techniques. In a first step, the client initiates the process of accessing a file by requesting a lookup or some equivalent operation at step 31. The request for such an operation transfers over the communications link 11 at step 32. The client 30, as known in NFS systems, then waits for a response. The server 10 performs the requested operation in step 33 and generates a return file handle 34. In this particular embodiment, and typically, the file handle 34 includes an FSID field 35 that identifies the particular file system, an inode number 36 that uniquely identifies the file designated by the client path name within the file system and a generation number 37 that provides a time history for each inode number. The resulting file handle returns during step 40 to the client 30 that, in step 41, receives the response and stores the file handle for subsequent use. Step 42 represents a subsequent process by which the client 30 issues a call using the file handle for transfer to the server 10 in step 43. Step 44 represents the conversion of the file handle to a form that identifies a specific file and the process of completing the designated operation. Thereafter the server 10 sends information over the communications link 11 during a transfer 45. The client 30 receives and processes the return in step 46.

(Column 4, lines 8-33)

An addressing scheme for file servers. The file handle in a call that accesses a file includes a file system identification, a file identification or inode number and a generation number. The addressing system converts

the multi-bit file identification number into an alternative path name that identifies a location for the file without the need for converting the real path name. In one particular embodiment the inode number in binary form is translated into a multiple digit hexadecimal form that is parsed into directory names. The last directory name provides the location of the designated file.

(Abstract)

FIG. 4 depicts, in a representative tree form, the first set of links 15 and the second set of links 21 incorporated in the file system 13. Each of these sets may be within the root directory or subdirectory thereunder as known in the art. By way of example, assume (1) that the file system identification is "FS1" (i.e., FSID=FS1), (2) that the client path name from a root or base directory is ABC/DEF/XYZ and (3) that the file handle returned to the client includes an inode number 1,000,000.sub.10. In accordance with this invention, when the server 13 receives the file handle, it converts the inode number into an equivalent hexadecimal value represented by the hexadecimal string "OF4240" that is an alternate path name. At 56 in FIG. 4, the system moves to the second set of links 21 and identifies the first directory "OF" within the possible range of directories "00" through "FF". At 57 and within the directory "OF", the system searches for the directory "42" and at 58 and within the directory "OF/42" searches for the entry "40". The entry "40" is interpreted as a file name because it is the last location in the alternate path name. This file points to a location from which the system determines the physical location of the file with inode 1,000,000 and the physical location of the appropriate entries in the second attribute table 22 shown in FIG. 1. Even though, as previously indicated, a six-digit hexadecimal number has the capacity of identifying 16,000,000 files, the location of any such file can be found, again assuming an even distribution of the files for any access, on the average in under 400 searches as opposed to 8,000,000 for a full file system sequential search. Utilizing the second set of links 21 limits the number of lookups to three, with each lookup being limited to a maximum of 256 entries. The time required for these lookups is insignificant in comparison with the lookup times tolerated with a first reference to a particular file in a client process.

(Column 4, line 66 to column 5, line 32)

In Figure 2 and the related description of column 4, lines 8-33, Pinkoski describes a single server 10, which receives a request from a client, generates a return file handle, returns the file handle to the client, receives a call using the file handle from the client, converts the file handle to a form that identifies a specific file, and sends the information to the client. In the abstract, Pinkoski describes an addressing scheme in which the file

server identified a file by a file handle that consists of a FSID field that identifies the particular file system, an inode number that uniquely identifies the file designated by the client path name within the file system and a generation number that provides a time history for each inode number. In column 3, lines 12-22, Pinkoski describes an alternate route to reach a file that remains in the same place. In column 5, lines 1-30, Pinkoski describes the information shown in Figure 4, which indicates the path that is identified in the file handle and more specifically the FSID. Nowhere in these sections or in any other section of Pinkoski is there a description of receiving a request for a referencing object from a client, wherein the referencing object refers to a referenced file system that has been moved to a location on a different server. In fact, Pinkoski clearly describes only one server as shown in Figure 2, which is the figure the present Office Action cites as teaching the movement of a file system to a different server. Thus, Pinkoski does not teach or suggest receiving a request for a referencing object from a client, wherein the referencing object refers to a referenced file system that has been moved to a location on a different server, as recited in claim 1, for example.

Though the present Office Action states the combination of Pinkoski and Kao teaches this feature, the Present Office Action did not cite a specific section of Kao as teaching this feature. Kao is directed to a virtual node architecture that is used to create a three-dimensional directory. A virtual node is created for each file and directory. A look-up procedure is used to find a specified file or directory name by sequentially searching the stack beginning with the top directory node and returning the node corresponding to the first occurrence of the name encountered in the search. However, even though Kao searches for files in a stack, Kao does not teach or suggest receiving a request for a referencing object from a client, wherein the referencing object refers to a referenced file system that has been moved to a location on a different server.

Moreover, neither reference teaches or suggests the desirability of incorporating the subject matter of the other reference. A proper prima facie case of obviousness cannot be established by combining the teachings of the prior art absent some teaching, incentive, or suggestion supporting the combination. In re Napier, 55 F.3d 610, 613, 34 U.S.P.Q.2d 1782, 1784 (Fed. Cir. 1995); In re Bond, 910 F.2d 831, 834, 15 U.S.P.Q.2d 1566, 1568 (Fed. Cir. 1990). That is, there is no motivation offered in either reference

for the alleged combination. The Office Action alleges that the motivation for the combination is "so as to enable to get the returning message for the alternative path name from the user in the network system" and "to have a system receiving the request for manipulating the selected nodes or files or objects' location in a different level of directory or structure or hierarchical file from the user of the computer network system." As discussed above, neither Kao nor Pinkoski teaches or suggests receiving a request for a referencing object from a client, wherein the referencing object refers to a referenced file system that has been moved to a location on a different server. Pinkoski discloses an alternate route to reach a file that remains in the same place. The Office Action acknowledges that Kao does not teach this feature. Neither reference teaches or suggests a referencing object that refers to a referenced file system that has been moved to a location on a different server. Thus, the only teaching or suggestion to even attempt the alleged combination is based on a prior knowledge of Applicants' claimed invention thereby constituting impermissible hindsight reconstruction using Applicants' own disclosure as a guide.

One of ordinary skill in the art, being presented only with Kao and Pinkoski, and without having a prior knowledge of Applicants' claimed invention, would not have found it obvious to combine and modify Kao and Pinkoski to arrive at Applicants' claimed invention. To the contrary, even if one were somehow motivated to combine Kao and Pinkoski, and it were somehow possible to combine the systems, the result would not be the invention, as recited in claim 1. The result would be to simply search for a file and provide alternate paths to get to the file. The resulting system still would not reference a file system that has been moved to a location on a different server.

In view of the above, Applicants respectfully submit that Kao and Pinkoski, taken alone or in combination, fail to teach or suggest the features of claims 1, 16, and 31. At least by virtue of their dependency on claims 1 and 16, the features of dependent claims 2-4, 7, 8, 17-19, 22, and 23 are not taught or suggested in Kao and Pinkoski, whether taken individually or in combination. Accordingly, Applicants respectfully request withdrawal of the rejection of claims 1-4, 7, 8, 16-19, 22, 23, and 31 under 35 U.S.C. § 103(a).

II. 35 U.S.C. § 103, Alleged Obviousness, Claims 5, 6, 9-12, 20, 21, and 24-27

The Final Office Action rejects claims 5, 6, 9-12, 20, 21, and 24-27 under 35 U.S.C. § 103(a) as being unpatentable over Kao (U.S. Patent No. 5,870,734) in view of Pinkoski (U.S. Patent No. 5,742,817) and further in view of Thompson et al. (U.S. Patent No. 5,463,772).

The deficiencies of the Kao and Pinkoski have been addressed above. That is, Kao and Pinkoski, taken alone or in combination, fail to teach or suggest “receiving a request for a referencing object from a client, wherein the referencing object refers to a referenced file system that has been moved to a location on a different server” (emphasis added). Thompson is directed to a Transparent Peripheral File System that includes a Peripheral File System Adapter which communicates with a host operating system at the vnode level of file operation by packaging such communications for transmission over an I/O system interface. Thus, Thompson does not provide for the deficiencies of Kao and Pinkoski. In that, Thompson does not teach or suggest receiving a request for a referencing object from a client, wherein the referencing object refers to a referenced file system that has been moved to a location on a different server.

In view of the above, Applicants respectfully submit that Kao, Pinkoski and Thompson, taken alone or in combination, fail to teach or suggest the features of claims 1, 16, and 31. At least by virtue of their dependency on claims 1 and 16, the features of dependent claims 5, 6, 9-12, 20, 21, and 24-27 are not taught or suggested in Kao, Pinkoski and Thompson, whether taken individually or in combination. Accordingly, Applicants respectfully request withdrawal of the rejection of claims 5, 6, 9-12, 20, 21, and 24-27 under 35 U.S.C. § 103(a).

III. 35 U.S.C. § 103, Alleged Obviousness, Claims 13-15, 28-30, and 32

The Final Office Action rejects claims 13-15, 28-30, and 32 under 35 U.S.C. § 103(a) as being unpatentable over Kao (U.S. Patent No. 5,870,734) in view of Thompson et al. (U.S. Patent No. 5,463,772). This rejection is respectfully traversed. Because this rejection is essentially the same as in the previous Office Action, this rejection is

respectfully traversed for the same reasons stated in the previous response filed August 13, 2004, the remarks of which are hereby incorporated by reference. The following remarks are provided in rebuttal to the statements in the Final Office Action beginning on page 2, section 3.

The Final Office Action states:

Applicants argued that, "Thompson discuss data compression that is not the same as encoding the data file system identifier." (Page 15, lines 11-13).

Thompson et al. Of 5,463,772 teaches data compression / decompression and also teaches a large number of algorithms that implement the properties of the desired file system by converting the v-node-style file operation including the converting of streams of character (col. 19, lines 2-30 and see abstract, col. 3, lines 40-55 and col. 4, lines 1-20).

In the previous response, Applicants respectfully submitted that, while it is true that the cited sections of Thompson discuss data compression on disk storage, it is submitted that data compression is not the same as encoding the file system identifier. Data compression, as the name implies, is performed on the data, the contents of a file, while encoding the file system identifier is performed on the identifier, or name, of the file.

Secondly, Applicants submitted that one of ordinary skill in the art would not equate compression with encoding, as these embody two different concepts. The online site, Webopedia, which bills itself as the number one online encyclopedia dedicated to computer technology, defines data compression as simply "[s]toring data in a format that requires less space than usual", noting that "[d]ata compression is particularly useful in communications because it enables devices to transmit or store the same amount of data in fewer bits." This site does not bother to define encoding, although the online Merriam Webster online dictionary defines the verb encode as "to convert (as information) from one system of communication into another; especially: to convert (a message) into code".

The present Office Action states that Thompson teaches data compression / decompression and also teaches a large number of algorithms that implement the properties of the desired file system by converting the v-node-style file operation including the converting of streams of character at column 19, lines 2-30, the abstract, column 3, lines 40-55 and column 4, lines 1-20, which read as follows:

The remote peripheral file system 17 converts the vnode-style file operation commands (and their associated streams of data) into appropriate sequences of drive mechanism oriented commands drawn from a (hardware) command set implemented by the internal machinery that is the real guts of the random access mass storage peripheral 3. This conversion is tailored for the specific hardware command set and drive parameters in force (e.g., size parameters—two drives might have the same basic command set but differ in the number of surfaces available) with the assistance of a collection of drivers 18. These drivers 18 also assist in the formatting of data coming from the mechanism into collections reflecting the file level of organization with which they were originally stored.

In accordance with those ends, the drivers 18 are functionally coupled to a peripheral command interpreter 19. Typically, interpreter 19 is also implemented in firmware, and it oversees the actual conversion between the programmatically (and digitally) conveyed/expressed commands or data and the actual analog signals that eventually inhabit the tangible peripheral mechanism. To this end the peripheral command interpreter 19 is typically assisted by a mechanism controller 20, which amounts to a fairly extensive chunk of circuitry. It is here that are found controllers for stepper motors and linear motors, etc. The mechanism controller 20 is electrically coupled to the actual peripheral mechanism (drive motors, heads, amplifiers, sensors, etc.) and its medium (or media) 21.

(Column 19, lines 2-29)

A Transparent Peripheral File System (TPFS) includes a Peripheral File System Adapter (PFSA) which communicates with a host operating system at the vnode level of file operation by packaging such communications for transmission over an I/O system interface, such as SCSI. A file peripheral system remote from the host and in a peripheral relation thereto is responsive to the PFSA, without an intervening file server. The peripheral file system produces hardware commands for the mass storage device whose space it manages. The peripheral file system may be embedded in a mass storage device, a lump in the interconnecting interface cable, or a smart interface card in the backplane of the host. The peripheral file system may include a daisy chain connection to allow the propagation of vnode communication to other peripheral file systems. In this way a hierarchy of peripheral file systems may be physically mounted to one another in a way that corresponds to how they are logically mounted. A Character To File Translator (CFX) allows a TPFS to emulate raw mode access, even if the host system is not equipped with a vnode file operation interface. CFX does this by converting to and from streams of characters communicated with an application using this ersatz raw mode, while instead of storing the data on the storage medium itself, actually relying upon the remote peripheral file system for storing, or having stored, the streams of characters as a file.

(Abstract)

The third conventional way of managing the storage of information on a mass storage medium is through the agency of a "file system." Files have names, and the file system itself keeps track of where on the medium a given file is, what kind it is, how big it is, etc. Ordinarily, the file system is an integral component implemented within the operating system. (However, in some older disc based operating systems supporting track/sector calls for disc operation, the file manager was an optional subsystem). Requests for file operations (i.e., requests for a list of files and their size and status, commands to read a file, etc.,) are placed with the operating system, which in turn invokes the file system. The file system has general knowledge about the architecture of the particular mass storage device(s) upon whose medium the content of the file system resides. Its overall task is to provide the algorithms that implement the properties of the desired file system. There are generally several powerful tools at its disposal to assist in that implementation. These generally include buffering, interrupt driven I/O, DMA, and a structured division of labor in the software that includes "software drivers." The drivers have detailed particular knowledge about the command set for the mass storage device being used. Thus, the file system may be viewed as a rather large collection of program modules whose collective purpose is to accept commands, file names and other parameters (such as data in R/W that is to become a file) as input and produce therefrom an appropriate sequence of "surface/track/sector talk" that also exploits buffering, interrupt driven I/O, and DMA.

(Column 3, lines 31-60)

Historically, there are good reasons why the file system has been located in the CPU and implemented as part of the operating system. In the beginning, there were no microprocessors, and processing power (i.e., the ability to execute a program) was at as much of a premium as memory was. Since a file system contains a large number of algorithms, only a processor executing a program could hope to implement any sort of reasonable file system. At the same time, embedded control systems using microprocessors were still a thing of the future. Users and their programs interacted with the operating system, which after due processing gave tasks to software drivers that in turn communicated with interface cards dedicated to the particular peripheral. A cable conveyed the final low-level hardware commands to a controller in the peripheral. As the various enhancements, such as buffered I/O, interrupt driven I/O, DMA, and caching were developed to improve performance, corresponding changes were incorporated into the operating system, as well as into the other software systems using those enhancements. In this way the range of

capabilities and the degree of performance of peripheral mass storage subsystems were both increased and optimized.

(Column 3, line 67 to column 4, line 21)

In column 19, lines 2-29, Thompson describes a remote peripheral file system that converts the vnode-style file operation commands (and their associated streams of data) into appropriate sequences of drive mechanism oriented commands drawn from a (hardware) command set implemented by the internal machinery that is the real guts of the random access mass storage peripheral. In the abstract, Thompson describes a peripheral file system that communicates with a host operating system at a vnode level of file operation by packaging such communications for transmission over an I/O system interface. A peripheral file system remote from the host and in a peripheral relation thereto is responsive to a peripheral file system adapter, without an intervening file server. In column 3, lines 31-60, Thompson describes management of the storage of information on a mass storage medium using a file system and that the file system may be viewed as a rather large collection of program modules whose collective purpose is to accept commands, file names and other parameters. In column 3, lines 67 to column 4, line 21, Thompson describes a file system that is located in a CPU and implemented as part of the operating system and that a file system contains a large number of algorithms, only a processor executing a program could hope to implement any sort of reasonable file system.

Kao and Thompson, taken alone or in combination fail to teach or suggest "receiving a request for a file system object, wherein the request includes an encoded file system identifier, which has been encoded using a predetermined, system wide encoding algorithm" (emphasis added). The present Office Action acknowledges that Kao does not teach encoding a file system identifier using a predetermined system wide encoding algorithm. Thompson teaches the conversion of vnode-style file operation commands (and their associated streams of data) into appropriate sequences of drive mechanism oriented commands drawn from a (hardware) command set implemented by the internal machinery that is the real guts of the random access mass storage peripheral. Thomson also teaches a Character-To-File (CFX) translator which is composed of a SCSI File System Driver that contains information

about the SFS command set, which SFS commands are coupled together, and how to encode/decode data sent to/from the TPFS (see column 25, lines 54-64 and column 29, lines 27-35). However, the CFX does not encoding a file system identifier using a predetermined system wide encoding algorithm as can be seen in Thompson's description of the Character-To-File (CFX) translator at column 25, line 54 to column 26, line18, which reads as follows:

Referring now to FIG. 7, software routine CFX 71 cooperates with portions of the Peripheral File System Adapter (PFSA) 13. There is a portion 72 thereof called SFSX (explained in the Appendices) that operates via path 75 in conjunction with the VFS 10 in the host 2. SFSX 72 is the vnode level interface for file system operations; the functionality SFSX provides has been central to many of the discussions in the Specification. SFSX cooperates with SFSD 73 (SCSI File System Driver, also explained in the Appendices).

What CFX 71 does is provide an alternate raw mode path for using SFSD 73, without need for SFSX 72. Note that this means that a user or their application can perform ersatz raw mode access to the storage device of a TPFS even if the host computer system 2 does not include a VFS! (Which would mean that the TPFS would otherwise be unusable--but the (ersatz) "raw mode" access to the storage device would still work!)

CFX 71 communicates with the using agency via path 76 (function calls or driver entry points and the associated passing of parameters). CFX 71 receives via path 76 requests for ersatz raw mode operation (e.g., OPEN, CLOSE, READ, WRITE, as well as others that could be device dependent) and the associated stream of characters. CFX 71 converts these requests into corresponding file operations (CREATE, LOOKUP, READ, WRITE, . . .) that are then passed to SFSD 73. CFX 71 will invent file names as needed. CFX 71 is also responsive to SFSD 73 for traffic in the direction of from SFSD to CFX, and sends to the using agency via path 76 character streams and associated status information.

In this section, Thompson describes a CFX that receives requests for ersatz raw mode operation (e.g., OPEN, CLOSE, READ, WRITE, as well as others that could be device dependent) and the associated stream of characters. The CFX converts these requests into corresponding file operations (CREATE, LOOKUP, READ, WRITE, . . .) that are then passed to SFSD which would encode the file operation. Thompson does not teach or suggest encoding a file system identifier using a predetermined system wide encoding algorithm.

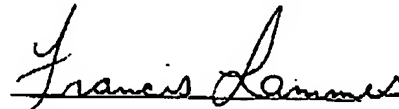
In view of the above, Applicants respectfully submit that Kao and Thompson, taken alone or in combination, fail to teach or suggest the features of claims 13, 28, and 32. At least by virtue of their dependency on claims 13 and 28, the features of dependent claims 14, 15, 29, and 30 are not taught or suggested in Kao and Thompson, whether taken individually or in combination. Accordingly, Applicants respectfully request withdrawal of the rejection of claims 13-15, 28-29, and 32 under 35 U.S.C. § 103(a).

IV. Conclusion

It is respectfully urged that the subject application is patentable over the prior art of record and is now in condition for allowance. The Examiner is invited to call the undersigned at the below-listed telephone number if in the opinion of the Examiner such a telephone conference would expedite or aid the prosecution and examination of this application.

Respectfully submitted,

DATE: March 17, 2005



Francis Lammes
Reg. No. 55,353
Yee & Associates, P.C.
P.O. Box 802333
Dallas, TX 75380
(972) 385-8777
Agent for Applicants